A. Kubernetes Architecture

1. Worker Node (Node)

Node –> Physical or virtual machine where containers are deployed.

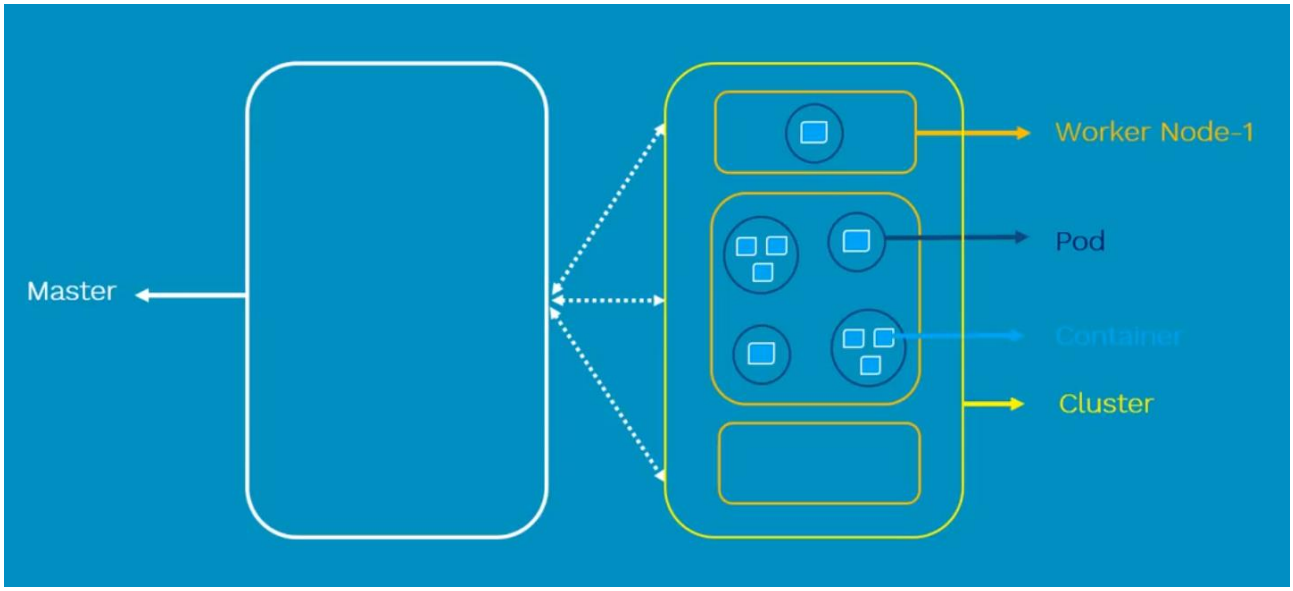
|

(One or more) Pods –> Wrapper around containers, we interact and manage containers through pods

|

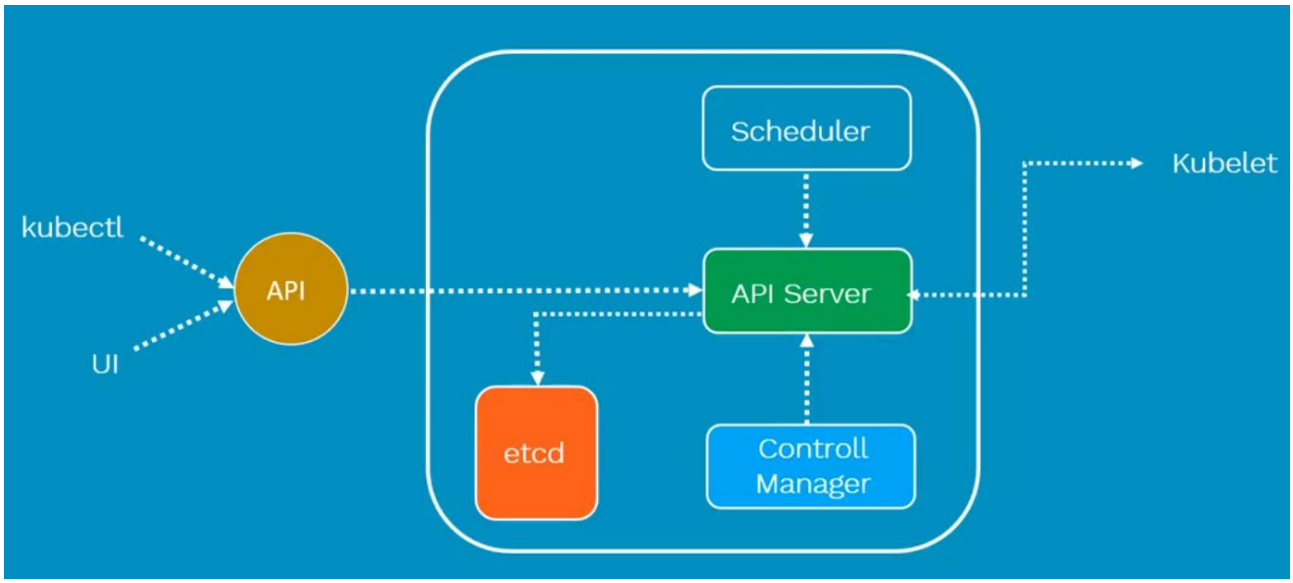
(One or more) Containers –> Runtime environment for containerized applications. Containers run microservices and are not ideal for monolithic applications.

Multiple worker nodes connected to the same network form a cluster



2. Master (consists of 4 components)

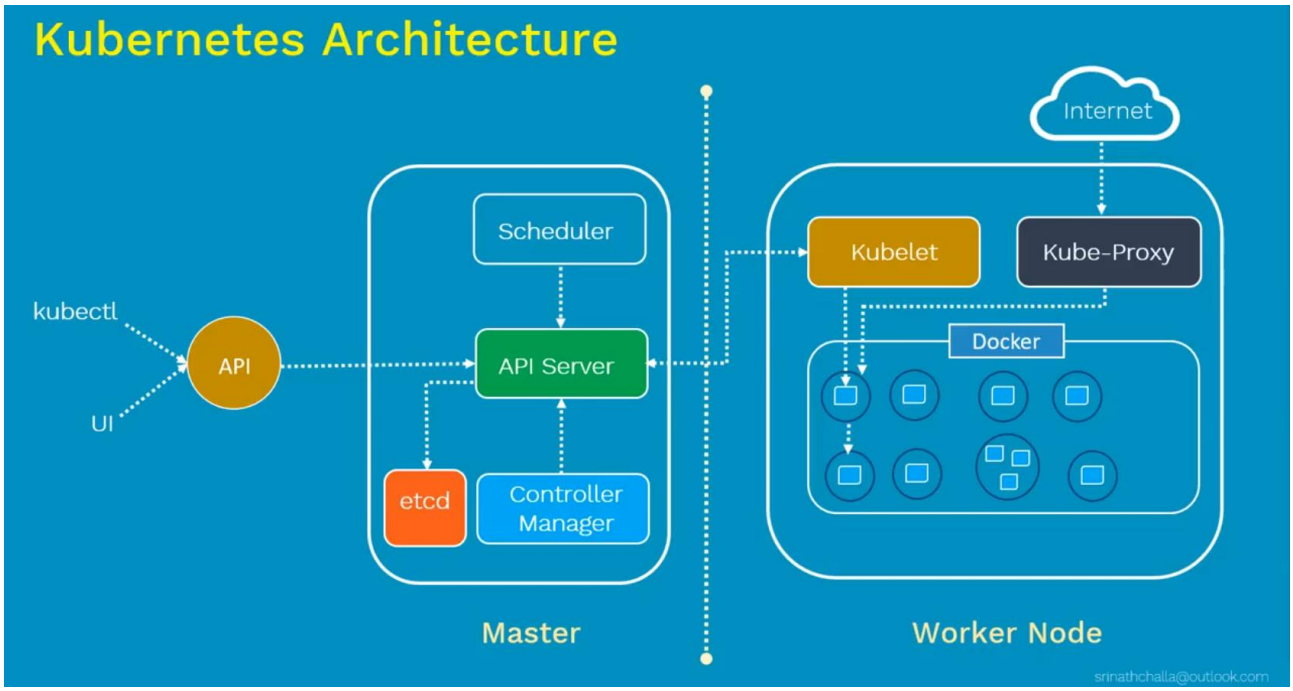
a. API server -> Gate keeper of the entire cluster. It validates and configures API objects (Pods, services, replication controllers, deployments etc). Kubectl and UI interact with the API.
b. Scheduler -> Physically schedules pods across multiple nodes based on the constraints mentioned in the configuration file.
c. Control Manager -> Includes Node controller, replication controller, endpoint controller, service and token controllers. These controllers are responsible for overall health of entire cluster.
d. etcd -> Distributed key-value database. It stores the current cluster state at any point of time. Any kubernetes component can query the etcd to understand the state of the cluster.



Some more components of Worker node:

Kubelet -> It runs on each worker node and monitors the health of that node. It ensures that containers described in the specification (fetched from API on Kubernetes master) are running and healthy.

Kube-proxy -> It is responsible for maintaining the internetwork of configuration. It maintains the distributed network across all the nodes, across all the pods and across all containers.

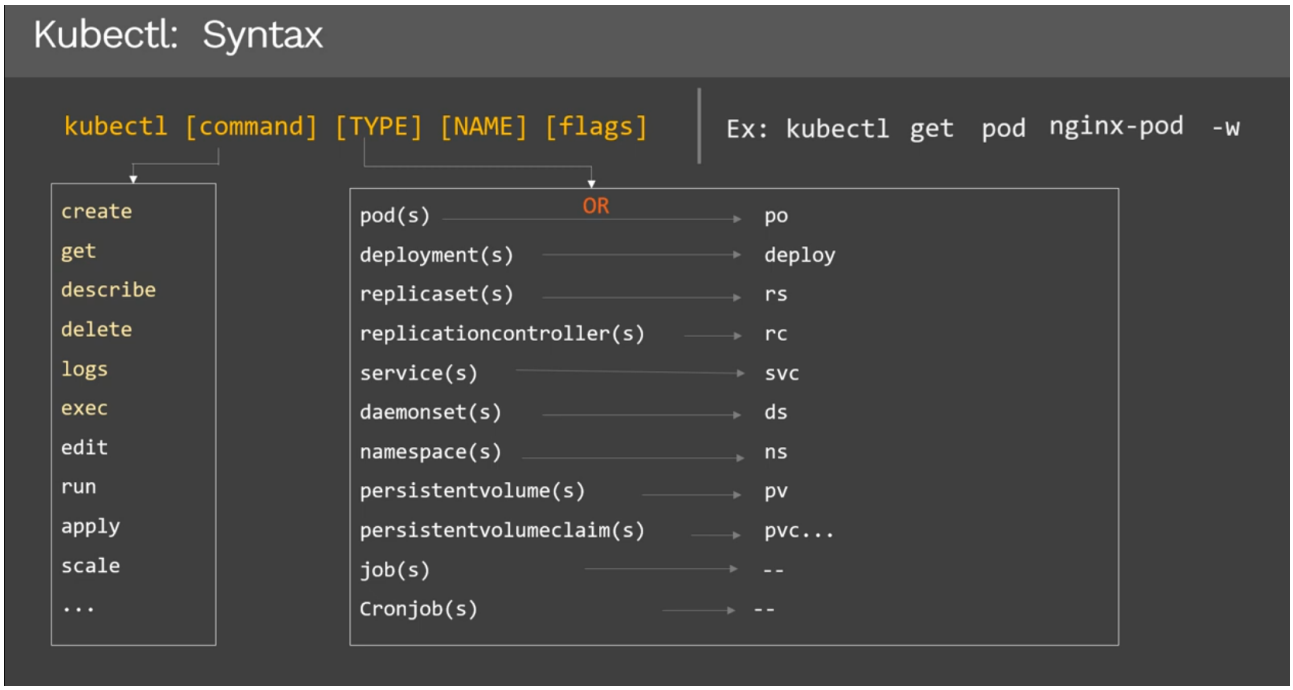


B. Different ways of installing Kubernetes:

1. Play-with-k8s: https://labs.play-with-k8s.com/
2. Minikube
3. Kubeadm
4. Googlr Kubernetes Engine (GKE)
5. Amazon EKS
6. Azure Kubernetes Service

C. Kubectl commands

Syntax: kubectl [command] [TYPE] [NAME] [flags]



1. Create:

To create a resource from a file

$ kubectl create -f pod-example.yaml

$ kubectl create -f deploy-example.yaml

To create a resource from multiple files in a directory

$ kubectl create -f <directory>

2. Get:

To display high level info of resources

kubectl get [TYPE(s)] [NAME(s)] [FLAGS] – List one or more resources

$ kubectl get pods (display all pods)

$ kubectl get pods <pod-name> (display info of a specific pod)

$ kubectl get pods -o wide

$ kubectl get pods,deploy

3. Describe:

To fetch complete details of resources

$ kubectl describe nodes <node-name>

$ kubectl describe pods <pod-name>

$ kubectl describe pods

4. Delete:

To delete resources

$ kubectl delete -f pod.yaml

$ kubectl delete pods,services -l name=<label-name>

$ kubectl delete pods –all

5. Exec:

To execute a command against a container in a pod.

$ kubectl exec <pod-name> date

$ kubectl exec <pod-name> -c <container-name> date

$ kubectl exec -it <pod-name> /bin/bash

6. Logs:

Print the logs for a container in a pod

$ kubectl logs <pod-name>

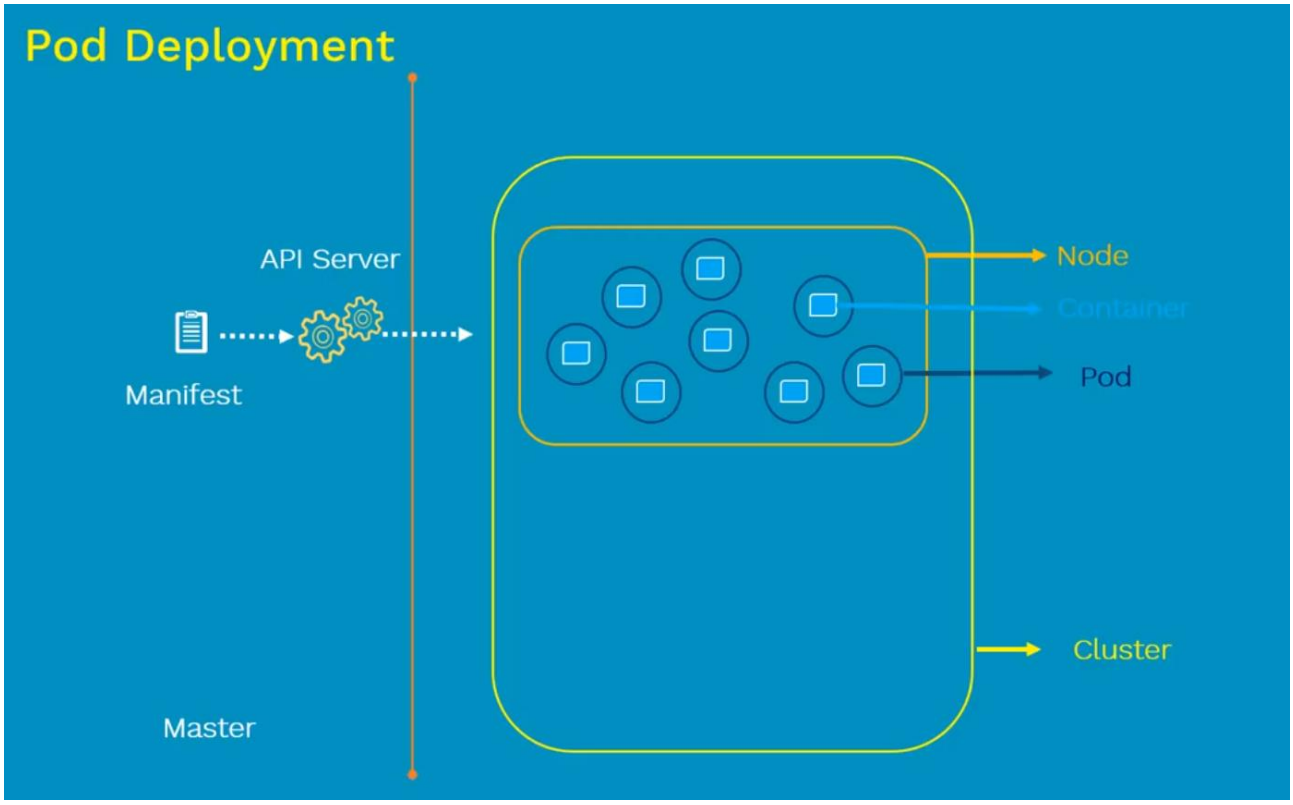
$ kubectl logs -f <pod-name>

D. Pods

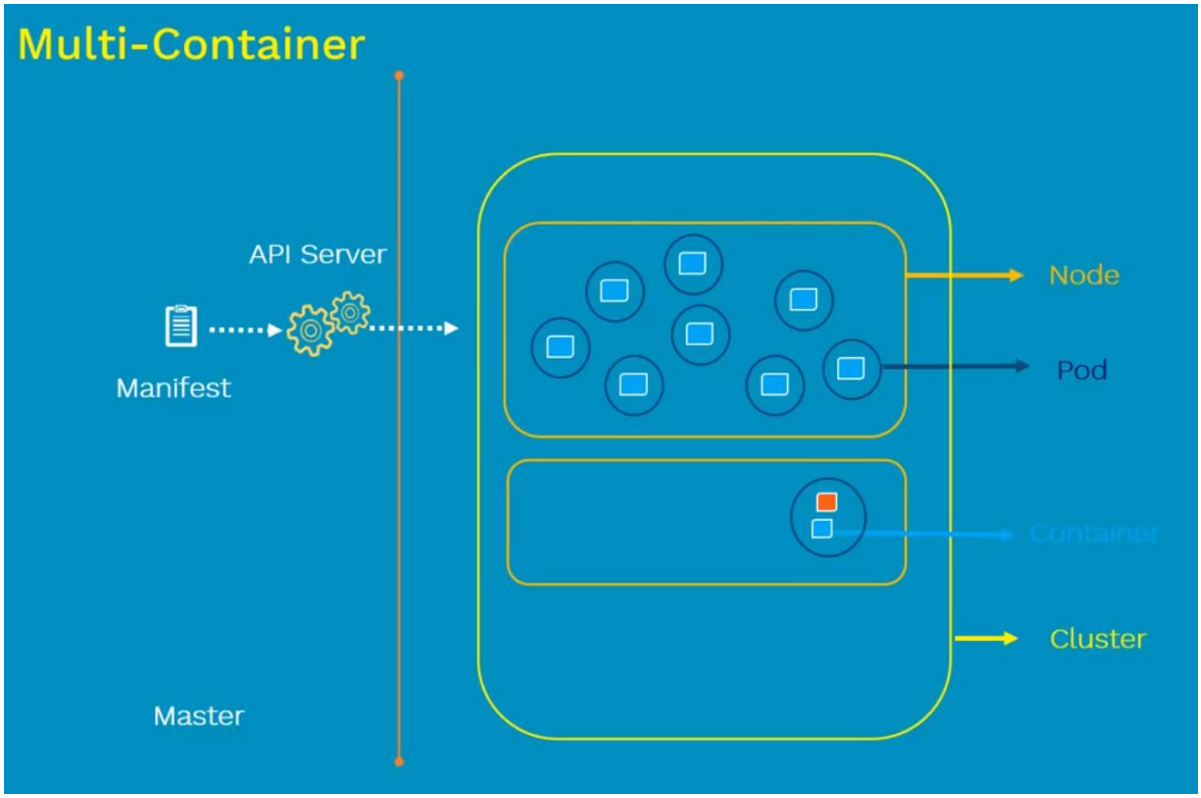
Pod is an atomic unit of scheduling in Kubernetes.

Pod deployment: Pod manifest file is written which contains container images that we wish to deploy and submit it to the API server on the master node. After that API server and scheduler components on master node decide and deploy these pods on appropriate worker nodes.

Each pod should contain only one container. To scale up an application, we replicate the pods to create multiple instances of the same app within the same node. In case where worker node has insufficient capacity to scale up, another node can be created which contains the same app instances (pods).



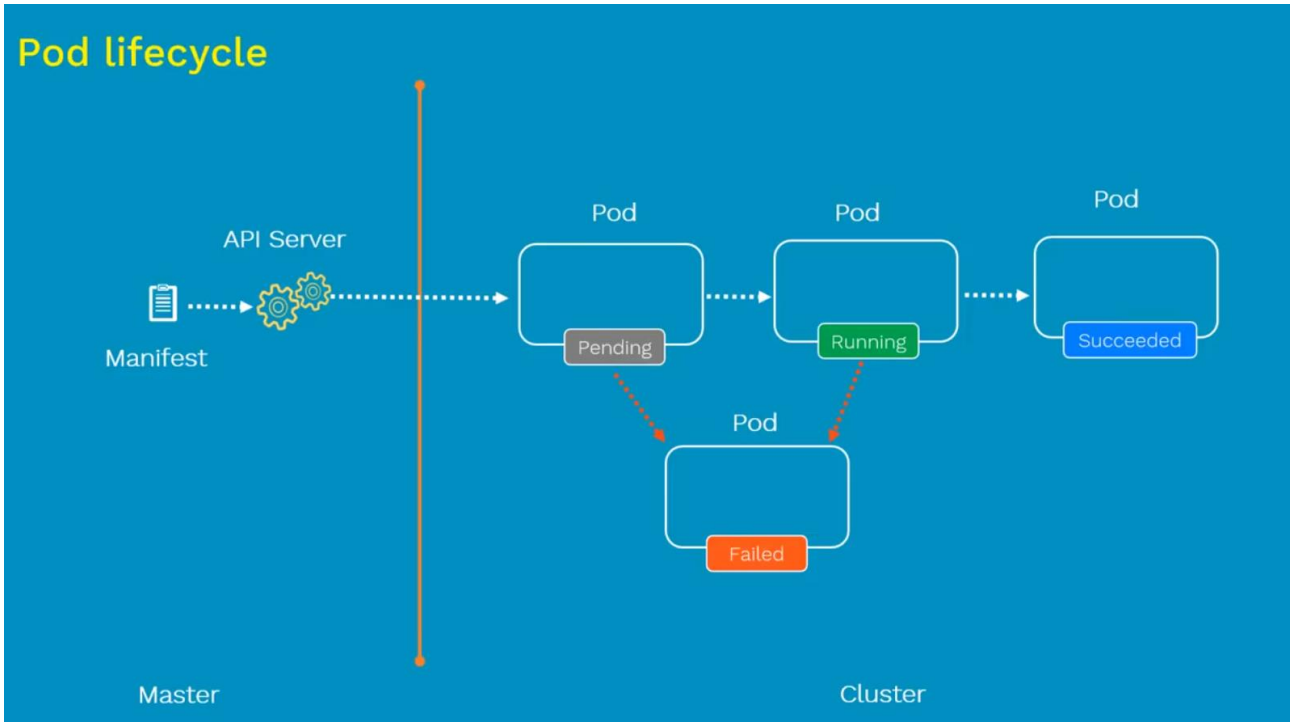
Multi-container pods: These are rarely used. Sometimes a container requires a helper container to perform certain processing. When the app container is created, the helper container also gets created and when the app container dies, the helper container also dies because they are part of the same pod. These two containers can communicate with each other using a localhost because they share the same network namespace plus they share the same storage space as well.



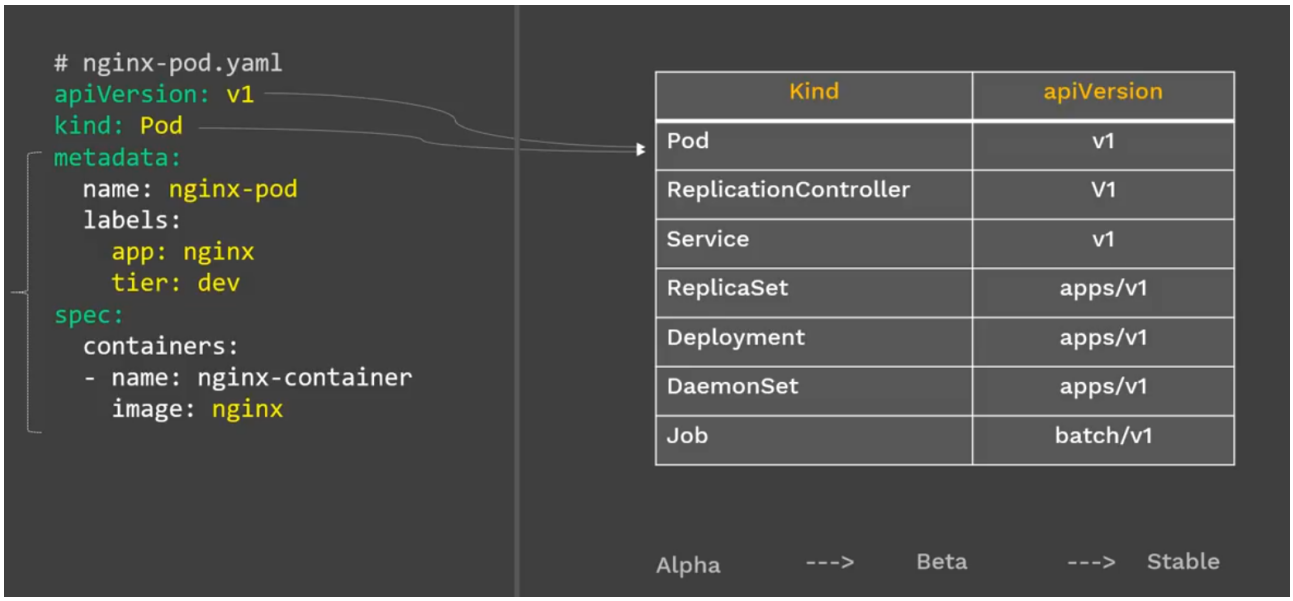
Pod lifecycle:

Write a config/manifest file and submit it to the API server on the cluster. It gets scheduled on a worker node on a Kubernetes cluster. Once it gets scheduled, it goes to the pending state.

1. Pending: In this state, node will download all the container images and start starting the containers. It stays in pending state until all containers are up and running.
2. Running: Once all containers are up and running, the main purpose of the pod is achieved.
3. Succeeded: After running, the pod is shutdown.
4. Failed: This stage can only be achieved from pending or running states. If for some reason the pod doesn't start, it moves to the failed state.



Pod Config/manifest:



Pod – Create & Display

```
[schalla@master ~]$ kubectl create -f nginx-pod.yaml
pod/nginx-pod created
```

```
[schalla@master ~]$ kubectl get pod
NAME        READY     STATUS    RESTARTS   AGE
nginx-pod   1/1       Running   0          2m
```

```
[schalla@master ~]$ kubectl get pod -o wide
NAME        READY     STATUS    RESTARTS   AGE       IP            NODE      NOMINATED NODE
nginx-pod   1/1       Running   0          8m        10.240.1.26   node1     <none>
```

```
[schalla@master ~]$ kubectl get pod nginx-pod -o yaml
apiVersion: v1
items:
- apiVersion: v1
.......
```

Pod - Describe

```
[schalla@master ~]$ kubectl describe pod nginx-pod       # Display all details and events of a pod
Name:               nginx-pod
Namespace:          default
Priority:           0
PriorityClassName:  <none>
Node:               node1/10.128.0.5
Start Time:         Tue, 28 Aug 2018 07:06:55 +0000
Labels:             app=nginx
                    tier=dev
Annotations:        <none>
Status:             Running
IP:                 10.240.1.26
Containers:
  nginx-container:
.........
Events:
  Type    Reason     Age   From               Message
  ----    ------     ----  ----               -------
  Normal  Scheduled  14m   default-scheduler  Successfully assigned default/nginx-pod to node1
  Normal  Pulling    14m   kubelet, node1     pulling image "nginx"
  Normal  Pulled     14m   kubelet, node1     Successfully pulled image "nginx"
  Normal  Created    14m   kubelet, node1     Created container
  Normal  Started    14m   kubelet, node1     Started container
```

Pod - Testing

```
[schalla@master ~]$ ping 10.240.1.26        #Pinging Container IP from master
PING 10.240.1.26 (10.240.1.26) 56(84) bytes of data.
64 bytes from 10.240.1.26: icmp_seq=1 ttl=63 time=0.387 ms
64 bytes from 10.240.1.26: icmp_seq=2 ttl=63 time=0.361 ms
64 bytes from 10.240.1.26: icmp_seq=3 ttl=63 time=0.246 ms
^C
--- 10.240.1.26 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 0.246/0.331/0.387/0.063 ms

[schalla@master ~]$ kubectl exec -it nginx-pod -- /bin/sh #Getting a shell to running cont
# hostname
nginx-pod
# exit

[schalla@master ~]$ kubectl delete pod nginx-pod
pod "nginx-pod" deleted
```

E. Replication Controller

Ensures that a specified number of pods are running at any time.

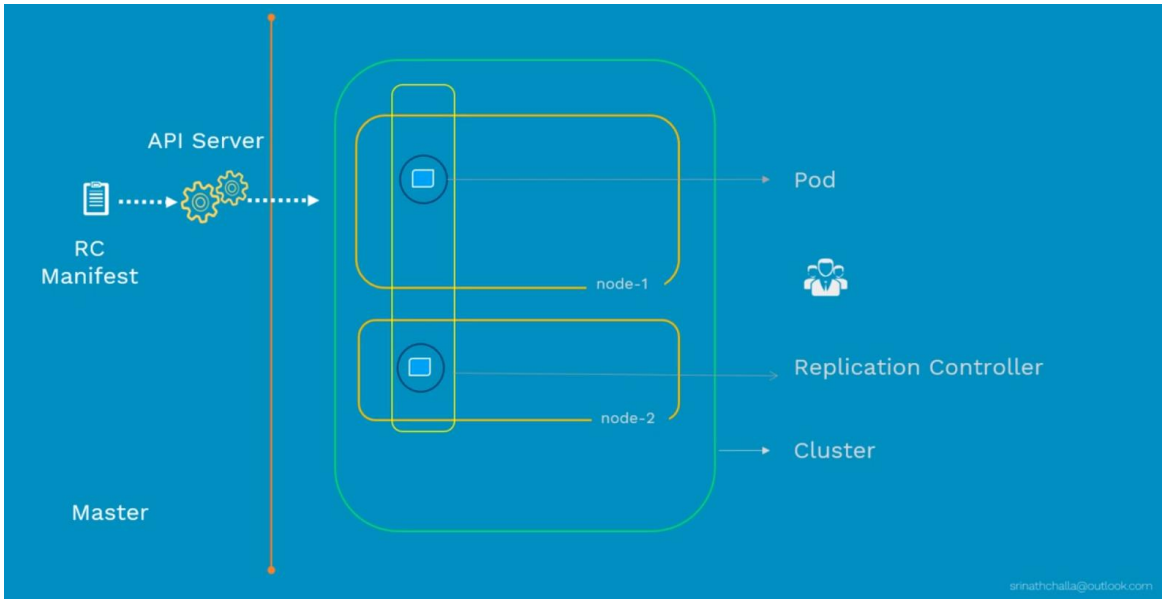
a) If there are excess pods, they get killed and vice versa.
b) New pods are launched when they fail, get deleted or terminated.

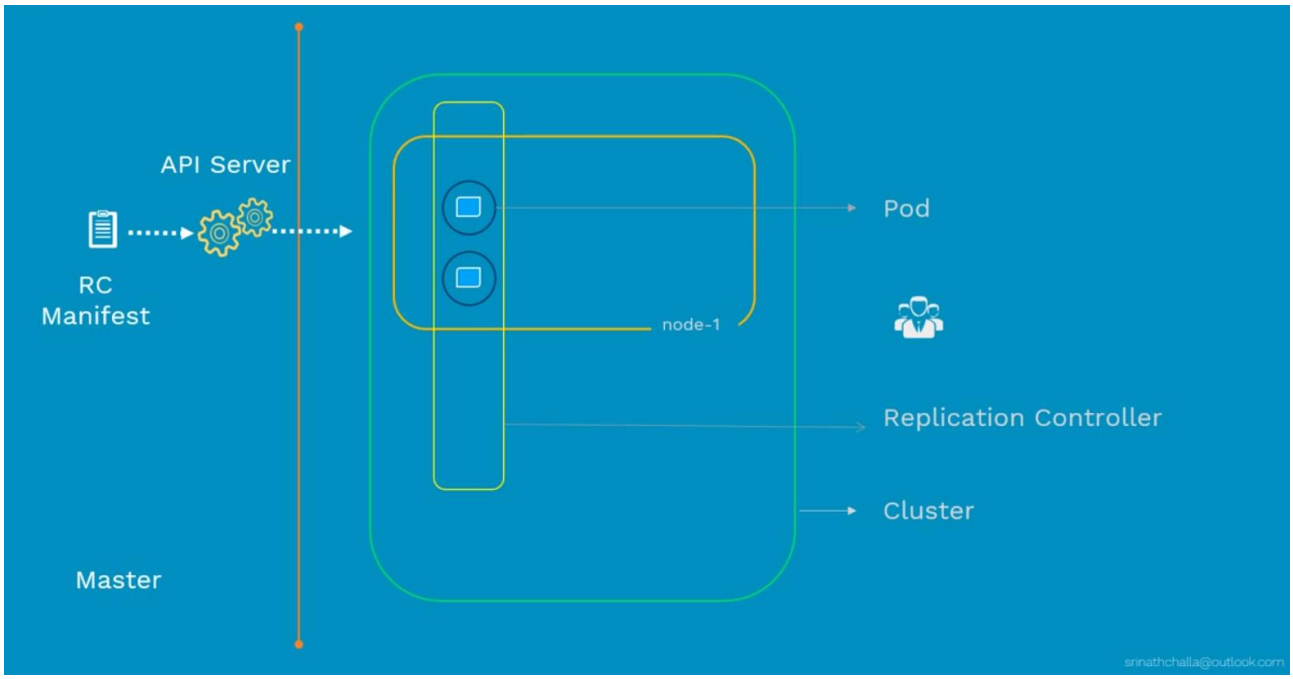
Replication controllers and pods are associated with “labels”.

Advantages:

1. High Availability

Assume you have two nodes running one instance each of the same pod. For some reason node 2 fails and the pod running inside it dies along with it., the control manager on the master node detects these changes and looks for a healthy node to create the deleted pod again within a short time.





Demo

Replication Controller – Config

```
# nginx-rc.yaml
apiVersion: v1
kind: ReplicationController
metadata:
  name: nginx-rc
spec:
  replicas: 3
  selector:
    app: nginx-app
  template:
    metadata:
      name: nginx-pod
      labels:
        app: nginx-app
    spec:
      containers:
      - name: nginx-container
        image: nginx
        ports:
        - containerPort: 80
```

Pod def

Replication Controller – Create & Display

```
[srinath@master ~]$ kubectl create -f nginx-rc.yaml
replicationcontroller/nginx-rc created
```

```
[srinath@master ~]$ kubectl get pods
NAME             READY     STATUS    RESTARTS   AGE
nginx-rc-62vpv   1/1       Running   0          6m
nginx-rc-fk67w   1/1       Running   0          6m
nginx-rc-qk4ph   1/1       Running   0          6m
```

```
[srinath@master ~]$ kubectl get po -l app=nginx-app
NAME             READY     STATUS    RESTARTS   AGE
nginx-rc-62vpv   1/1       Running   0          4m
nginx-rc-fk67w   1/1       Running   0          4m
nginx-rc-qk4ph   1/1       Running   0          4m
```

Replication Controller – Describe

```
[srinath@master ~]$  kubectl describe rc nginx-rc
Name:         nginx-rc
Namespace:    default
Selector:     app=nginx-app
Labels:       app=nginx-app
Annotations:  <none>
Replicas:     3 current / 3 desired
Pods Status:  3 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:  app=nginx-app
  Containers:
   nginx-container:
    Image:        nginx
    Port:         80/TCP
    Host Port:    0/TCP
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Events:
  Type    Reason            Age   From                    Message
  ----    ------            ----  ----                    -------
  Normal  SuccessfulCreate  25m   replication-controller  Created pod: nginx-rc-62vpv
  Normal  SuccessfulCreate  25m   replication-controller  Created pod: nginx-rc-fk67w
  Normal  SuccessfulCreate  25m   replication-controller  Created pod: nginx-rc-qk4ph
```

Replication Controller – Node Fail

```
[srinath@master ~]$ kubectl get po -o wide
NAME              READY     STATUS    RESTARTS   AGE       IP             NODE      NOMINATED NODE
nginx-rc-5lb6b    1/1       Running   0          16s       10.240.1.32    node1     <none>
nginx-rc-l64js    1/1       Running   0          16s       10.240.1.33    node1     <none>
nginx-rc-qrqv7    1/1       Running   0          16s       10.240.2.28    node2     <none>
```

```
[srinath@master ~]$ kubectl get nodes
NAME      STATUS     ROLES     AGE       VERSION
master    Ready      master    1d        v1.11.2
node1     Ready      <none>    1d        v1.11.2
node2     NotReady   <none>    1d        v1.11.2
```

```
[srinath@master ~]$ kubectl get po -o wide
NAME              READY     STATUS    RESTARTS   AGE       IP             NODE      NOMINATED NODE
nginx-rc-5lb6b    1/1       Running   0          11m       10.240.1.32    node1     <none>
nginx-rc-l64js    1/1       Running   0          11m       10.240.1.33    node1     <none>
nginx-rc-mtwzs    1/1       Running   0          2m        10.240.1.34    node1     <none>
nginx-rc-qrqv7    1/1       Unknown   0          11m       10.240.2.28    node2     <none>
```

srinathchalla@outlook.com

Replication Controller – Scaling up

```
[srinath@master ~]$ kubectl scale rc nginx-rc --replicas=5
replicationcontroller/nginx-rc scaled
```

```
[srinath@master ~]$ kubectl get rc nginx-rc
NAME       DESIRED   CURRENT   READY     AGE
nginx-rc   5         5         5         9m
```

```
[srinath@master ~]$ kubectl get po -o wide
NAME              READY     STATUS    RESTARTS   AGE       IP              NODE      NOMINATED NODE
nginx-rc-2x2kg    1/1       Running   0          36m       10.240.1.173    node1     <none>
nginx-rc-7kvhl    1/1       Running   0          36m       10.240.2.3      node2     <none>
nginx-rc-g7mwq    1/1       Running   0          33m       10.240.2.42     node2     <none>
nginx-rc-jgt28    1/1       Running   0          33m       10.240.1.3      node1     <none>
nginx-rc-wvmrx    1/1       Running   0          36m       10.240.2.4      node2     <none>
```

Replication Controller – Scaling down

```
[srinath@master ~]$ kubectl scale rc nginx-rc --replicas=3
replicationcontroller/nginx-rc scaled
```

```
[srinath@master ~]$ kubectl get rc nginx-rc
NAME       DESIRED   CURRENT   READY     AGE
nginx-rc   3         3         3         45m
```

```
[srinath@master ~]$ kubectl get po -o wide
```

Replication Controller – Delete

```
[srinath@master ~]$ kubectl delete -f nginx-rc.yaml  #kubectl delete rc nginx-rc
replicationcontroller "nginx-rc" deleted
```

```
[srinath@master ~]$ kubectl get rc
No resources found.
```

```
[srinath@master ~]$ kubectl get po -l app=nginx-app
No resources found.
```

Replication controllers are old, ReplicaSets are the new standard.

F. ReplicaSets

Ensures that a specified number of pods are running at any time.

a) If there are excess pods, they get killed and vice versa.
b) New pods are launched when they fail, get deleted or terminated.

Replication controllers and pods are associated with “labels”.

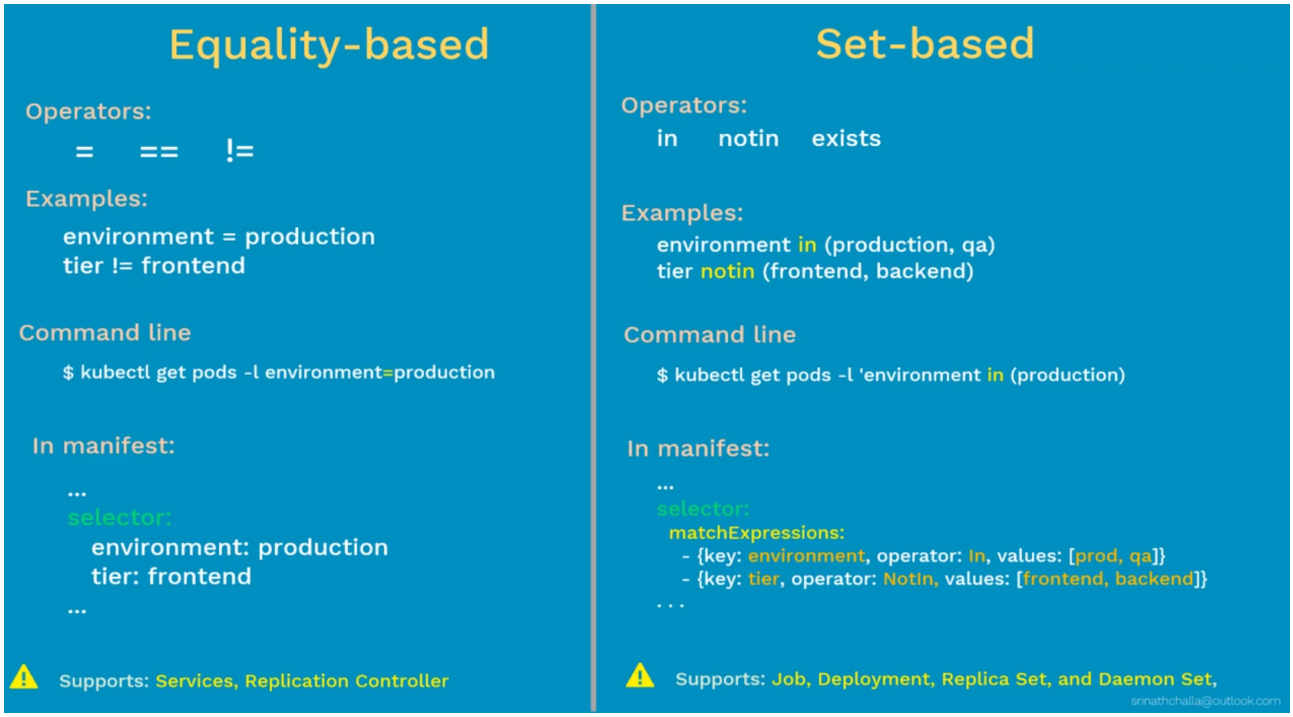
ReplicaSet is next-gen replication conrtroller.

Replication Controller -> Equality-based selectors

ReplicaSet -> Set based selectors

Labels & Selectors:

We define labels under metadata of pod’s config-file. Labels are key-value pairs that are generally attached to pods. Controllers and services manage these pods using selectors. There are two types of selectors – Equality based and set based. Equality based selectors are old hence set based selectors are practiced often.



Older resources like replication controllers and services don’t need ‘matchLabels’ field in their manifest file for selectors.

‘matchLabels’ supports on newer resources such as replicasets, deployments, jobs, daemonsets.

Demo:

ReplicaSet– Manifest file

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: nginx-rs
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx-app
    matchExpressions:
      - {key: tier, operator: In, values: [frontend]}
  template:
    metadata:
      name: nginx-pod
      labels:
        app: nginx-app
        tier: frontend
    spec:
      containers:
      - name: nginx-container
        image: nginx
        ports:
        - containerPort: 80
```

ReplicaSet– Create & Display

```
[srinath@master ~]$ kubectl create -f nginx-rs.yaml
replicaset/nginx-rs created
```

```
[srinath@master ~]$ kubectl get pods
NAME             READY     STATUS    RESTARTS   AGE
nginx-rs-62vpv   1/1       Running   0          6m
nginx-rs-fk67w   1/1       Running   0          6m
nginx-rs-qk4ph   1/1       Running   0          6m
```

```
[srinath@master ~]$ kubectl get po -l tier=frontend
NAME             READY     STATUS    RESTARTS   AGE
nginx-rs-62vpv   1/1       Running   0          4m
nginx-rs-fk67w   1/1       Running   0          4m
nginx-rs-qk4ph   1/1       Running   0          4m
```

ReplicaSet– Describe

```
srinath@master:$ kubectl get rs nginx-rs -o wide
NAME       DESIRED   CURRENT   READY     AGE       CONTAINERS        IMAGES    SELECTOR
nginx-rs   3         3         3         14s       nginx-container   nginx     app=nginx-app,tier in (frontend)
```

```
srinath@master:$ kubectl describe rs nginx-rs
Name:         nginx-rs
Namespace:    default
Selector:     app=nginx-app
Labels:       app=nginx-app
Annotations:  <none>
Replicas:     3 current / 3 desired
Pods Status:  3 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:  app=nginx-app
  Containers:
   nginx-container:
    Image:        nginx
    Port:         80/TCP
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Events:
  Type    Reason            Age   From                   Message
  ----    ------            ----  ----                   -------
  Normal  SuccessfulCreate  8m    replicaset-controller  Created pod: nginx-rs-8vcfq
  Normal  SuccessfulCreate  8m    replicaset-controller  Created pod: nginx-rs-l2t5z
  Normal  SuccessfulCreate  8m    replicaset-controller  Created pod: nginx-rs-br776
```


ReplicaSet – Scheduling

```
[srinath@master ~]$ kubectl get po -o wide
NAME             READY     STATUS    RESTARTS   AGE       IP             NODE      NOMINATED NODE
nginx-rc-5lb6b   1/1       Running   0          16s       10.240.1.32    node1     <none>
nginx-rc-l64js   1/1       Running   0          16s       10.240.1.33    node1     <none>
nginx-rc-qrqv7   1/1       Running   0          16s       10.240.2.28    node2     <none>
```

```
[srinath@master ~]$ kubectl get nodes
NAME      STATUS     ROLES     AGE       VERSION
master    Ready      master    1d        v1.11.2
node1     Ready      <none>    1d        v1.11.2
node2     NotReady   <none>    1d        v1.11.2
```

```
[srinath@master ~]$ kubectl get po -o wide
NAME             READY     STATUS    RESTARTS   AGE       IP             NODE      NOMINATED NODE
nginx-rc-5lb6b   1/1       Running   0          11m       10.240.1.32    node1     <none>
nginx-rc-l64js   1/1       Running   0          11m       10.240.1.33    node1     <none>
nginx-rc-mtwzs   1/1       Running   0          2m        10.240.1.34    node1     <none>
nginx-rc-qrqv7   1/1       Unknown   0          11m       10.240.2.28    node2     <none>
```

ReplicaSet – Scaling up

```
[srinath@master ~]$ kubectl scale rs nginx-rs --replicas=5
replicationcontroller/nginx-rc scaled
```

```
[srinath@master ~]$ kubectl get rs nginx-rc
NAME       DESIRED   CURRENT   READY     AGE
nginx-rc   5         5         5         9m
```

```
[srinath@master ~]$ kubectl get po -o wide
NAME             READY     STATUS    RESTARTS   AGE       IP             NODE  NOMINATED NODE
nginx-rc-2x2kg   1/1       Running   0          36m       10.240.1.173   node1 <none>
nginx-rc-7kvhl   1/1       Running   0          36m       10.240.2.3     node2 <none>
nginx-rc-g7mwq   1/1       Running   0          33m       10.240.2.42    node2 <none>
nginx-rc-jgt28   1/1       Running   0          33m       10.240.1.3     node1 <none>
nginx-rc-wvmrx   1/1       Running   0          36m       10.240.2.4     node2 <none>
```

ReplicaSet – Scaling down

```
[srinath@master ~]$ kubectl scale rs nginx-rs --replicas=3
replicationcontroller/nginx-rc scaled
```

```
[srinath@master ~]$ kubectl get rs nginx-rcs
NAME       DESIRED   CURRENT   READY     AGE
nginx-rc   3         3         3         45m
```

```
[srinath@master ~]$ kubectl get po -o wide
NAME             READY     STATUS    RESTARTS   AGE       IP             NODE  NOMINATED NODE
nginx-rc-2x2kg   1/1       Running   0          36m       10.240.1.173   node1 <none>
nginx-rc-jgt28   1/1       Running   0          33m       10.240.1.3     node1 <none>
nginx-rc-wvmrx   1/1       Running   0          36m       10.240.2.4     node2 <none>
```

ReplicaSet – Delete

```
[srinath@master ~]$ kubectl delete -f nginx-rc.yaml  #kubectl delete rc nginx-rc
replicationcontroller "nginx-rc" deleted
```

```
[srinath@master ~]$ kubectl get po -l app=nginx-app
No resources found.
```

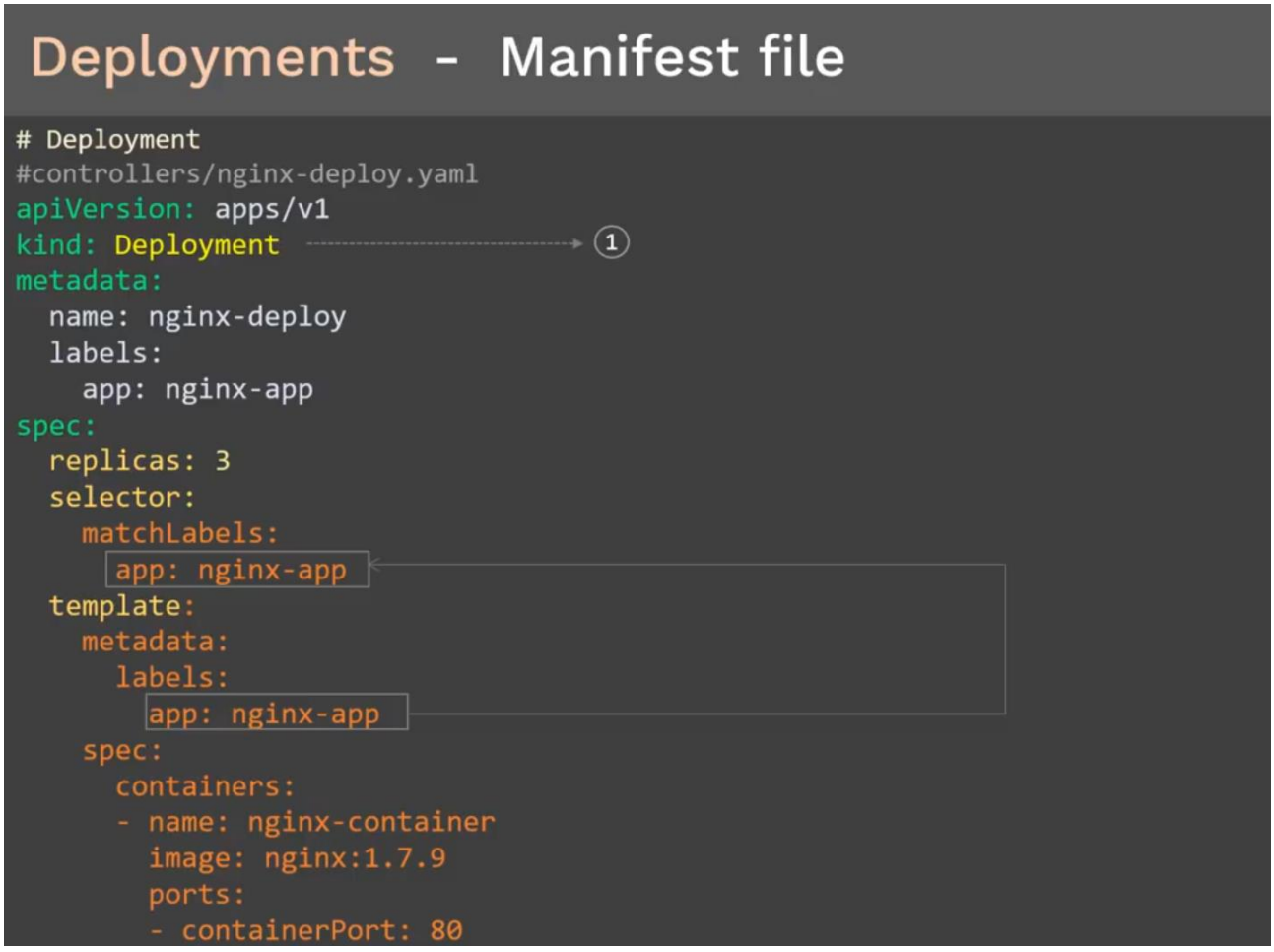
G. Deployments

Deployment is a controller. Deployment is all about updates and rollbacks.

Features:

1. Multiple Replicas: Using deployment you can create multiple replicas of pods for high availability and load balancing.
2. Upgrade: 4 strategies –
   - Recreate (switching from version A to version B with a downtime)
   - Rolling update (Ramped or incremental – used by Kubernetes as default method)
   - Canary (gradually shifting production traffic from version A to version B)
   - Blue/Green (version B which is green is deployed alongside version A which is blue with exactly same amount of instances. The traffic is shifted from version A to version B at the load balancer level. Advantage: instant rollout and rollback)
3. Rollback: If something goes wrong with the current upgrade then, deployment controller will allow you to rollback to previous stable upgrade.
4. Scale up or scale down: Once you deploy the application, you can scale up or scale down the application instances based on the load. For this, update the replica field in deployment manifest file accordingly.
5. Pause and resume: You can pause the deployment process in progress as and when needed. Used to test and validate new versions of the feature.

Demo



Deployments – Create & Display

```
[srinath@master ~]$  kubectl create -f nginx-deploy.yaml
deployment.apps/nginx-deployment created
```

```
[srinath@master ~]$ kubectl get deploy -l app=nginx-app
NAME               DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   3         3         3            3           1m
```

```
[srinath@master ~]$ kubectl get rs -l app=nginx-app
NAME                          DESIRED   CURRENT   READY     AGE
nginx-deployment-c75f4bb64    3         3         3         3m
```

```
[srinath@master ~]$ kubectl get po -l app=nginx-app
NAME                                READY     STATUS    RESTARTS   AGE
nginx-deployment-c75f4bb64-h7hph    1/1       Running   0          8m
nginx-deployment-c75f4bb64-pbmj4    1/1       Running   0          8m
nginx-deployment-c75f4bb64-sr4vt    1/1       Running   0          8m
```

Deployments – Describe

```
[srinath@master ~]$ kubectl describe deploy nginx-deployment

Name:                   nginx-deployment
Namespace:              default
CreationTimestamp:      Wed, 29 Aug 2018 11:39:31 +0000
Labels:                 app=nginx-app
Annotations:            deployment.kubernetes.io/revision=1
Selector:               app=nginx-app
Replicas:               3 desired | 3 updated | 3 total | 3 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=nginx-app
  Containers:
   nginx-container:
    Image:        nginx:1.7.9
    Port:         80/TCP
...
...
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
OldReplicaSets:  <none>
NewReplicaSet:   nginx-deployment-c75f4bb64 (3/3 replicas created)
Events:
  Type    Reason             Age   From                   Message
  ----    ------             ----  ----                   -------
  Normal  ScalingReplicaSet  13m   deployment-controller  Scaled up replica set nginx-deployment-c75f4bb64 to 3
```

Deployments – Update Deployment

```
[srinath@master ~]$ kubectl set image deploy nginx-deployment nginx-container=nginx:1.9.1
deployment.extensions/nginx-deployment image updated
```

```
[srinath@master ~]$ kubectl edit deploy nginx-deployment
deployment.extensions/nginx-deployment image updated
```

```
[srinath@master ~]$ kubectl rollout status deployment/nginx-deployment
Waiting for deployment "nginx-deployment" rollout to finish: 1 old replicas are pending termination...
Waiting for deployment "nginx-deployment" rollout to finish: 1 old replicas are pending termination...
deployment "nginx-deployment" successfully rolled out
```

```
[srinath@master ~]$ kubectl get deploy
NAME               DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   3         3         3            3           48m
```

srinathchalla@outlook.com

Deployments – Rollback Deployment

```
[srinath@master ~]$ kubectl set image deploy nginx-deployment nginx-container=nginx:1.91 --record
deployment.extensions/nginx-deployment image updated
```

```
[srinath@master ~]$ kubectl rollout status deployment/nginx-deployment
Waiting for deployment "nginx-deployment" rollout to finish: 1 out of 3 new replicas have been updated...
```

```
[srinath@master ~]$ kubect rollout history deployment/nginx-deployment
deployments "nginx-deployment"
REVISION  CHANGE-CAUSE
1  kubectl create --filename=nginx-deploy.yaml --record=true
2  kubectl set image deploy nginx-deployment nginx-container=nginx:1.91 --record=true
```

```
[srinath@master ~]$ kubectl rollout undo deployment/nginx-deployment
deployment.extensions/nginx-deployment
```

```
[srinath@master ~]$ kubectl rollout status deployment/nginx-deployment
deployment "nginx-deployment" successfully rolled out
```

srinathchalla@outlook.com

Deployments – Scaling up

```
[srinath@master ~]$ kubectl scale deployment nginx-deployment --replicas=5
deployment.extensions/nginx-deployment scaled
```

```
[srinath@master ~]$ kubectl get deploy
NAME               DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   5         5         5            5           20m
```

```
[srinath@master ~]$ kubectl get po
NAME                                READY     STATUS    RESTARTS   AGE
nginx-deployment-c75f4bb64-8dcmw    1/1       Running   0          3m
nginx-deployment-c75f4bb64-8dlb9    1/1       Running   0          21m
nginx-deployment-c75f4bb64-fxlrw    1/1       Running   0          3m
nginx-deployment-c75f4bb64-kxvtx    1/1       Running   0          3m
nginx-deployment-c75f4bb64-r6pjl    1/1       Running   0          3m
```


Deployments – Scaling down

```
[srinath@master ~]$ kubectl scale deployment nginx-deployment --replicas=1
deployment.extensions/nginx-deployment scaled
```

```
[srinath@master ~]$ kubectl get deploy
NAME               DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   1         1         1            1           24m
```

```
[srinath@master ~]$  kubectl get po -l app=nginx-app
NAME                                READY     STATUS    RESTARTS   AGE
nginx-deployment-c75f4bb64-h7hph    1/1       Running   0          8m
```

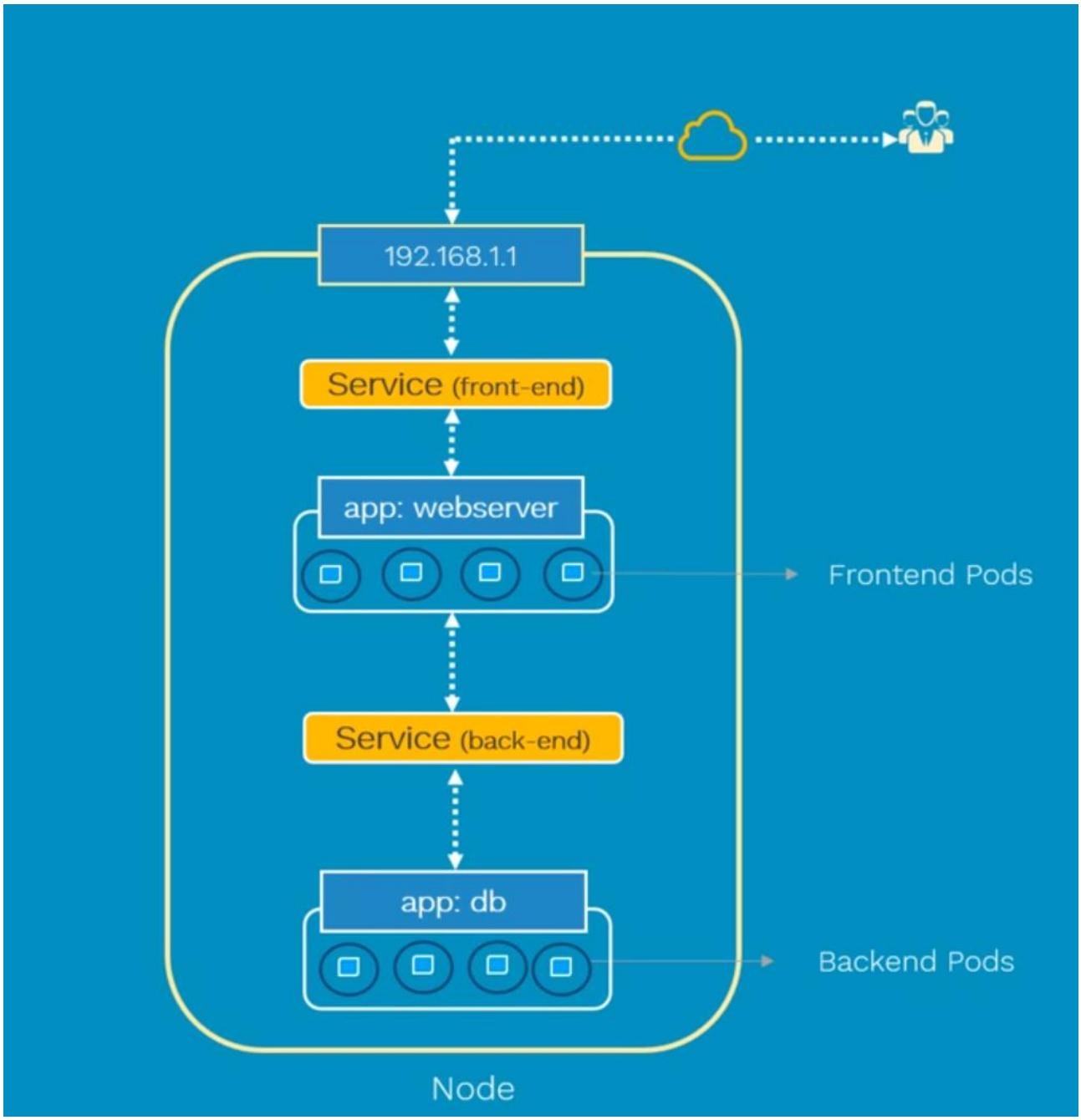

Deployments – Delete

```
[srinath@master ~]$ kubectl delete -f nginx-deploy.yaml
deployment.apps "nginx-deployment" deleted
```

```
[srinath@master ~]$ kubectl get po -l app=nginx-app
No resources found
```

H. Services

Service is a way of grouping of pods that running on the cluster. Services are cheap and you can have as many services as possible within your cluster. Services provide some of the important features that are standardized across the cluster such as load balancing, service discovery between apps.



Types of services:

1. Cluster IP service: It is reachable only within the cluster. The scope of cluster IP is confined to the cluster
2. NodePort: When you want to expose an app to the outside world on the internet.
3. LoadBalancer: To distribute incoming traffic over the available nodes to avoid overloading certain nodes and leaving others unused.

I. Volumes

Pods are ephemeral and stateless.

Volumes bring persistence to pods.

Adv of Kubernetes volumes vs Docker volumes:

- Container has access to volumes
- Volumes are associated with lifecycle of pod whereas in docker they are associated with lifecycle of a container.
- Kubernetes Supports multiple types of volumes
- Kubernetes volumes are much more mature and robust.

Volume categories:

1. Ephemeral: Same lifetime as pods
2. Durable: Beyond pods lifetime

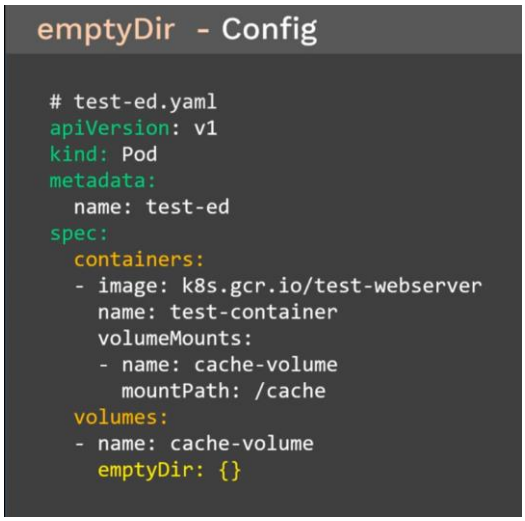
Volume types:

1. configMap
2. emptyDir
3. gcePersistentDisk
4. hostPath
5. persistentVolumeClaim
6. secret

emptyDir:

- Creates empty directory first created when a Pod is assigned to a Node
- Stays as long as pod is running
- Once pod is removed from a node, emptyDir is deleted forever.

Use cases: Temporary space



emptyDir – Create & Display

```
[srinath@master ~]$  kubectl create -f test-ed.yaml
pod/test-ed created
```

```
[srinath@master ~]$ kubectl get po
NAME              READY     STATUS    RESTARTS   AGE
test-ed           1/1       Running   0          49s
```

```
[srinath@master ~]$ kubectl exec test-ed df /cache
Filesystem     1K-blocks    Used Available Use% Mounted on
/dev/sda1       10474496 5389664   5084832  52% /cache
```

emptyDir – Describe

```
[srinath@master ~]$  kubectl describe pod test-ed
Name:               test-ed
Namespace:          default
Priority:           0
PriorityClassName:  <none>
Node:               node2/10.128.0.7
Start Time:         Thu, 30 Aug 2018 13:39:17 +0000
Labels:             <none>
Annotations:        <none>
Status:             Running
IP:                 10.240.2.162
Containers:
  redis-container:
    Container ID:   docker://bb2c13c35729e303428e30fe4aa06e724d1b9ef78677187af41082d763aab705
    Image:          redis
    ...
    Mounts:
      /cache from cache-volume (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-mdxv9 (ro)
    ...
Volumes:
  cache-volume:
    Type:    EmptyDir (a temporary directory that shares a pod's lifetime)
    ...
```

srinathchalla@outlook.com

emptyDir – Delete

```
[srinath@master ~]$    kubectl delete po test-ed
 pod "test-ed" deleted
```

hostPath:

- Mounts a file or directory from the host node's filesystem into your Pod.
- Remains even after the pod is terminated.
- Similar to docker volume.
- Host issues might cause problem to hostPath

hostPath – Config

```yaml
# HostPath
apiVersion: v1
kind: Pod
metadata:
  name: redis-hostpath
spec:
  containers:
    - name: redis-container
      image: redis
      volumeMounts:
      - mountPath: /test-mnt
        name: test-vol
  volumes:
  - name: test-vol
    hostPath:
      path: /test-vol
```

HostPath – Create and Display

```
[srinath@master ~]$  kubectl create -f redis-hostpath.yaml
pod/redis-hostpath created
```

```
[srinath@master ~]$ kubectl get po
NAME             READY     STATUS    RESTARTS   AGE       IP             NODE      NOMINATED NODE
redis-hostpath   1/1       Running   0          19m       10.240.2.163   node2     <none>
```

```
[srinath@master ~]$ kubectl exec redis-hostpath df /test-mnt
Filesystem     1K-blocks    Used Available Use% Mounted on
/dev/sda1       10474496 5389740   5084756  52% /test-mnt
```

HostPath – From Host to Pod

Host(node2):

```
[root@node2 ~]#  cd /test-vol
[root@node2 test-vol]# echo "From Host" > from-host.txt
[root@node2 test-vol]# cat from-host.txt
From Host
```

Pod:

```
[srinath@master ~]$ kubectl exec redis-hostpath cat /test-mnt/from-host.txt
From Host
```

HostPath – From Pod to Host

From Pod:

```
[srinath@master ~]$ kubectl exec redis-hostpath -it -- /bin/sh
# cd /test-mnt
# echo "From Pod" > from-pod.txt
# cat from-pod.txt
From Pod
# exit
```

From Host (node-2):

```
[root@node2 ~]# cd /test-vol
[root@node2 test-vol]# ls
from-pod.txt  from-host.txt
[root@node2 test-vol]# cat from-pod.txt
From Pod
```

HostPath – Delete

```
[srinath@master ~]$ kubectl delete po redis-hostpath
pod "redis-hostpath" deleted
```

```
[srinath@master ~]$ kubectl get po
No resources found
```

```
[root@node2 ~]# ls /test-vol
from-pod.txt  from-host.txt
```

gcePersistentDisk:

- Volume mounts a Google Compute Engine (GCE) Persistent Disk into Pod.
- Volume data is persisted after pod's termination.
- Read-write only on one node and Read-only on many nodes.

J. Persistent Volumes

Persistent volume subsystem provides a standard API for developers and administrators that abstract the details of how storage is provided from how it is consumed.

Persistent volume (PV) – Piece of storage in a cluster

Persistent Volume Claim (PVC) – Request for storage. Generally developers make this request along with read/write permissions.

Lifecycle of a Persistent Volume:

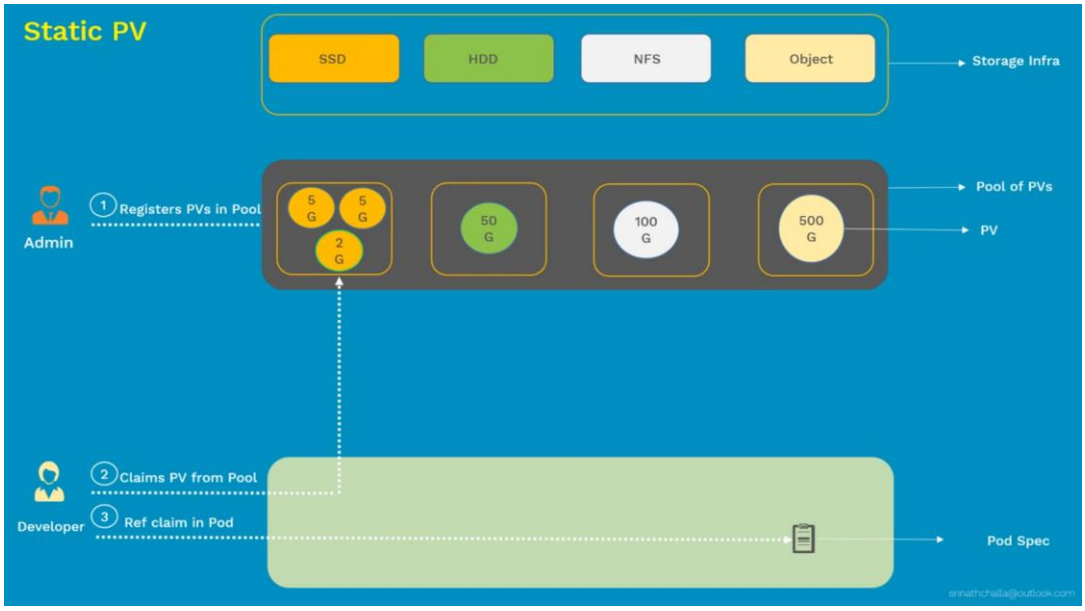
1. Provisioning – Administrator creates the storage volumes. These volumes can be any storage type such as block, NFS or distributed.
2. Binding – We bind the storage request to the persistent volume that was provisioned in earlier stage. This storage request in Kubernetes is called PVC.
3. Usage – Once bounded, then Kubernetes creates the pod and application can start using the volume in the pod.
4. Reclaim – Once a user/developer is done using the volume, they can delete the PVC from Kubernetes which allows reclaiming of resources.

There are two ways to provision volumes

1. Static Provisioning: Admin needs to create PV before developer requests for volume using PVC

Process:

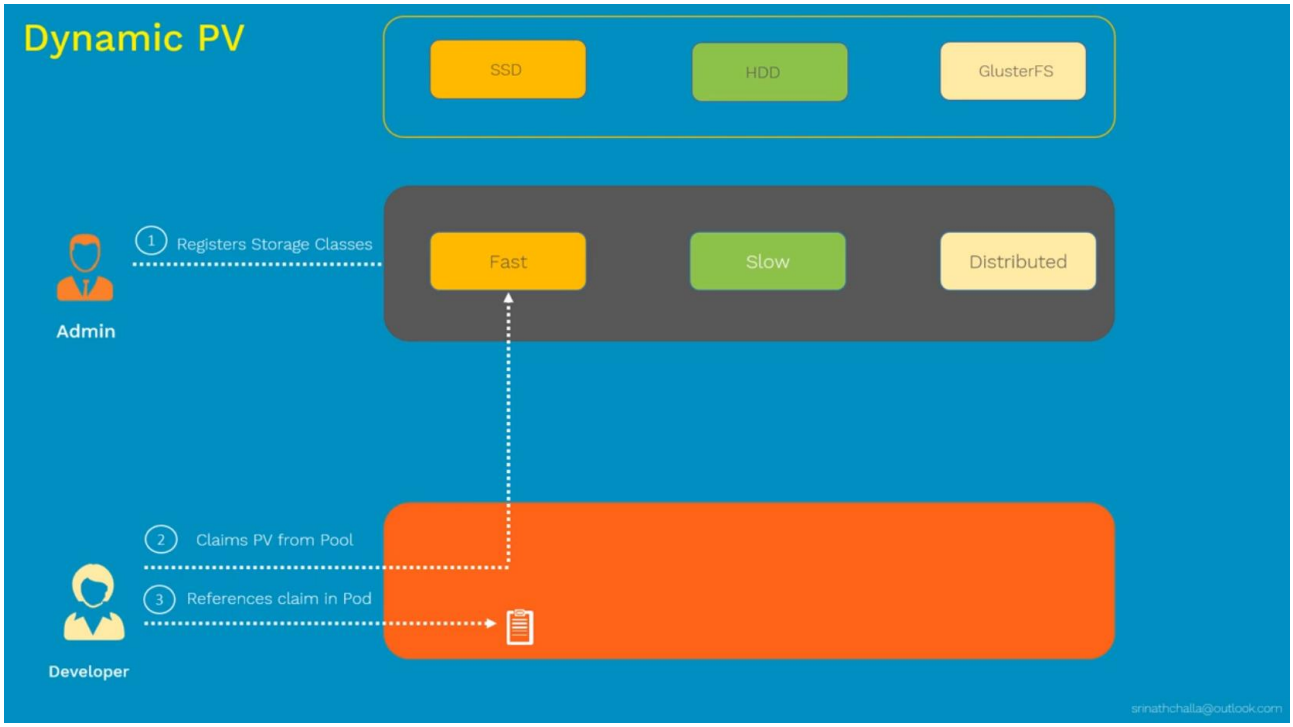
Storage infrastructure has different types of storage. Admin creates Storage chunks (PVs) of different capacities. Developer can create PVCs to request for a PV from this pool. The PVC gets bound with the requested PV. Developer can use this bounded PV in the pod. But in case when size of requested storage does not match with one in the available pool, developer has to wait till admin creates the right size PV.



2. Dynamic Provisioning: When developer creates a storage request, it simultaneously provisions the PV and binds it together.

Process:

Assume you have a storage infrastructure with different types of storage as shown in the image. We don't create the PVs manually instead we create something known as storage classes. Storage classes are a tag given to a storage based on the type of medium in the backend. These storage classes are created and maintained by admins and it is a one-time process. Developers need not worry about the availability of correct size of PV in the pool, the only thing developers need to make sure is the availability of the type of storage they need. Once that is confirmed, a developer can create a PVC which in turn creates a respective PV and it gets bounded. Once PVC gets bound to the PV, developer uses the claim inside the pod.



K. ConfigMap

Configuring containerized application:

- Container images are built to be portable
- Containers expect configuration from
    - Configuration files
    - Command line arguments
    - Environment variables

in any of the following format: INI, XML, JSON, custom format.

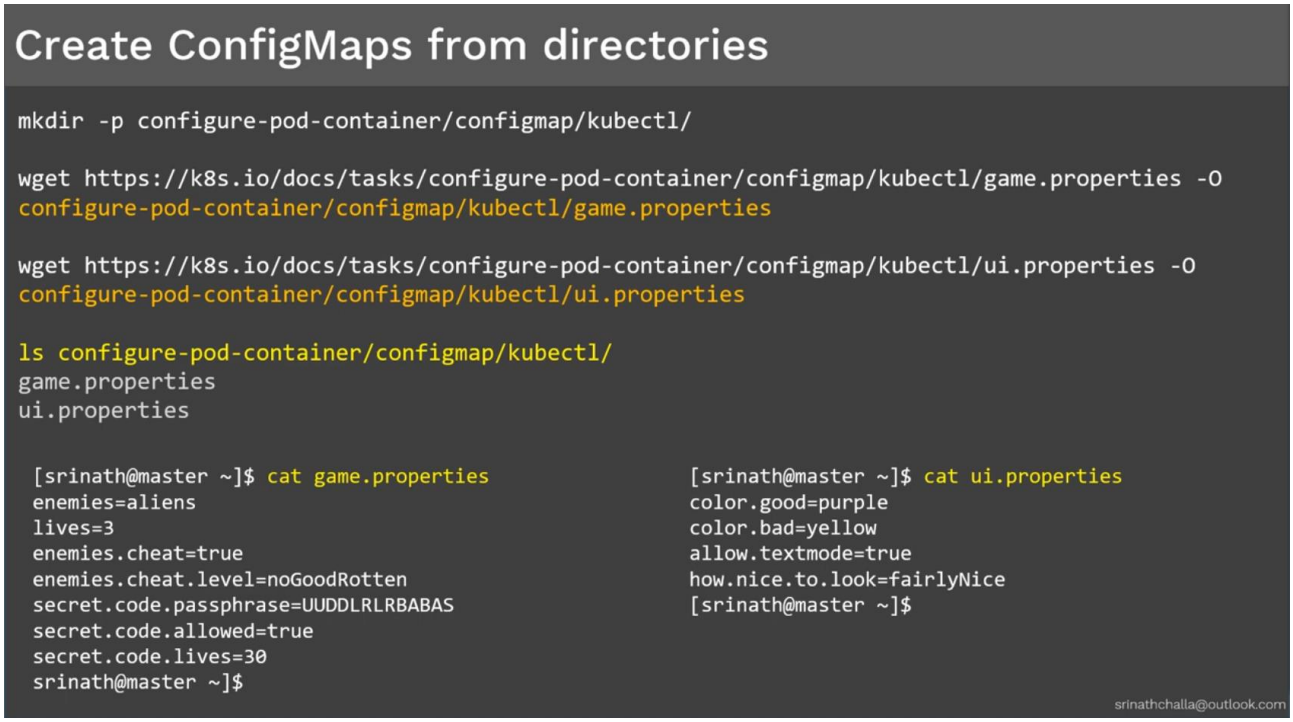
ConfigMaps:

- Decouples configuration from pods and components
- Stores configuration data as key-value pairs
    - Configuration files
    - Command line arguments
    - Environment variables
- Similar to secrets but don't contain sensitive information
- You must create a configmap before referencing it in a pod spec.

ConfigMaps can be created from directories, files, literals

Syntax to create a configMap: kubectl create configmap <map-name> <data-source>

- Path to dir/file: --from file
- Key-value pair: --from-literal



Create ConfigMaps from directories

```
srinath@master:$ kubectl get configmaps game-config -o yaml
apiVersion: v1
data:
  game.properties: |-
    enemies=aliens
    lives=3
    enemies.cheat=true
    enemies.cheat.level=noGoodRotten
    secret.code.passphrase=UUDDLRLRBABAS
    secret.code.allowed=true
    secret.code.lives=30
  ui.properties: |
    color.good=purple
    color.bad=yellow
    allow.textmode=true
    how.nice.to.look=fairlyNice
kind: ConfigMap
metadata:
  creationTimestamp: 2018-09-01T09:21:42Z
  name: game-config
  namespace: default
  resourceVersion: "598823"
  selfLink: /api/v1/namespaces/default/configmaps/game-config
  uid: 707531b3-adc8-11e8-b3de-42010a800003
```

Creating ConfigMaps from-file

```
srinath@master:$ curl -OL https://k8s.io/examples/pods/config/redis-config

% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   185  100   185    0     0    239      0 --:--:-- --:--:-- --:--:--   239
10043  100    43    0     0     29      0  0:00:01  0:00:01 --:--:--   132
```

```
srinath@master:$ cat redis-config

maxmemory 2mb
maxmemory-policy allkeys-lru
```

```
srinath@master:$ kubectl create configmap example-redis-config --from-file=redis-config

configmap/example-redis-config created
```

Accessing ConfigMaps in Pods

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: redis
spec:
  containers:
  - name: redis
    image: kubernetes/redis:v1
    volumeMounts:
    - mountPath: /redis-master
      name: config
  volumes:
    - name: config
      configMap:
        name: example-redis-config
        items:
        - key: redis-config
          path: redis.conf
```

Create ConfigMaps from literal values

```
srinath@master:$ kubectl create configmap special-config --from-literal=special.how=very

configmap/special-config created
```

```
srinath@master:$  kubectl get configmaps

 NAME             DATA      AGE
 special-config   2         2m
```

Create ConfigMaps from literal values

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: test-pod
spec:
  containers:
  - name: test-container
    image: k8s.gcr.io/busybox
    command: [ "/bin/sh", "-c", "env" ]
    env:
      - name: SPECIAL_LEVEL_KEY
        valueFrom:
          configMapKeyRef:
            name: special-config
            key: special.how
  restartPolicy: Never
```

```
srinath@master:$ kubectl logs test-pod | grep SPECIAL
SPECIAL_LEVEL_KEY=very
```

L. Ingress

Ingress exposes HTTP and HTTPS routes from outside the cluster to services within the cluster. Traffic routing is controlled by rules defined on the Ingress resource.

Here is a simple example where an Ingress sends all its traffic to one Service:

An Ingress may be configured to give Services externally-reachable URLs, load balance traffic, terminate SSL / TLS, and offer name-based virtual hosting. An Ingress controller is responsible for fulfilling the Ingress, usually with a load balancer, though it may also configure your edge router or additional frontends to help handle the traffic.

An Ingress does not expose arbitrary ports or protocols. Exposing services other than HTTP and HTTPS to the internet typically uses a service of type Service.Type-NodePort or Service.Type=LoadBalancer.

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: minimal-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - http:
      paths:
      - path: /testpath
        pathType: Prefix
        backend:
          service:
            name: test
            port:
              number: 80
```

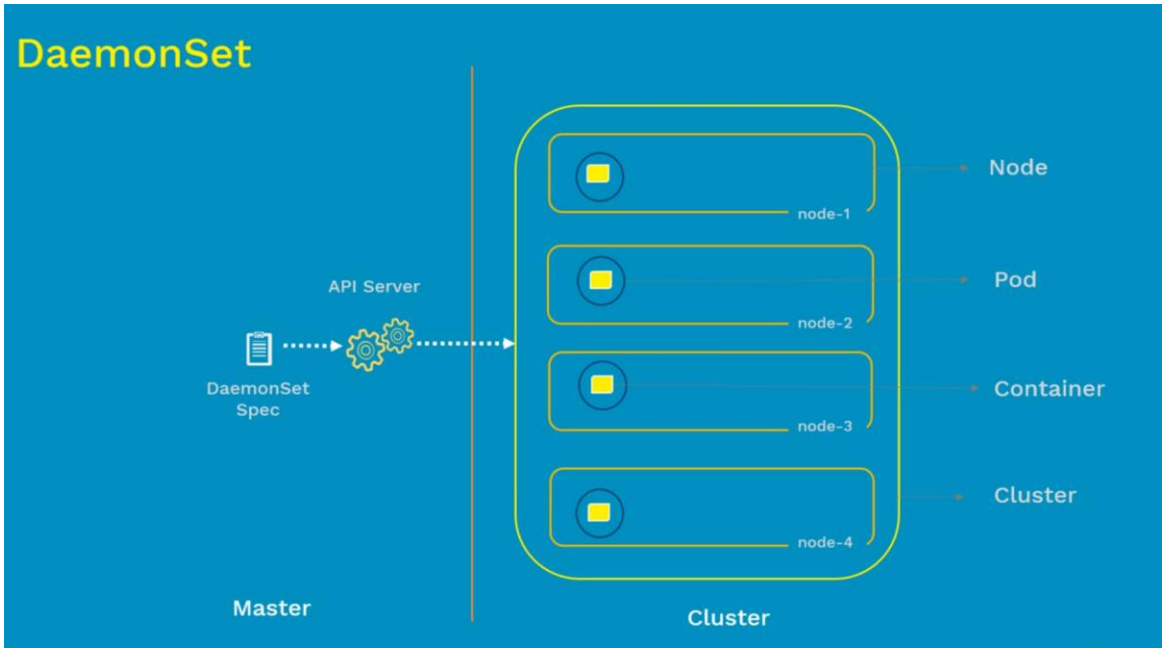
M. DaemonSet

- A DaemonSet ensures that all or some Nodes run a copy of a pod.

- As nodes are added to a cluster, pods are added.

- As nodes are removed from the cluster, pods are garbage collected.

- Deleting a DaemonSet will clean up the pods it created.



Use cases:

- Node monitoring daemons – ex: collectd
- Log collection daemons – ex: fluentd
- Storage daemons – ex: ceph

Demo:

DaemonSet – Config file

```
#fluentds.yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluentd-ds
spec:
  template:
    metadata:
      labels:
        name: fluentd
    spec:
      containers:
      - name: fluentd
        image: gcr.io/google-containers/fluentd-elasticsearch:1.20
  selector:
```

Pod

DaemonSet – Create and Display

```
[srinath@master ~]$  kubectl get no
NAME      STATUS    ROLES     AGE       VERSION
master    Ready     master    2d        v1.11.2
node1     Ready     <none>    2d        v1.11.2
node2     Ready     <none>    2d        v1.11.2
```

```
[srinath@master ~]$ kubectl create -f fluentd-daemonset.yaml
daemonset.apps/fluentd-ds created
```

```
[srinath@master ~]$ kubectl get po -o wide
NAME              READY     STATUS    RESTARTS   AGE       IP             NODE      NOMINATED NODE
fluentd-ds-22jhn  1/1       Running   0          58s       10.240.1.54    node1     <none>
fluentd-ds-2krwx  1/1       Running   0          58s       10.240.2.158   node2     <none>
```

```
[srinath@master ~]$ kubectl get ds
NAME         DESIRED   CURRENT   READY     UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE
fluentd-ds   2         2         2         2            2           <none>          3m
```

DaemonSet – Describe

```
[srinath@master ~]$  kubectl describe ds fluentd-ds
Name:           fluentd-ds
Selector:       name=fluentd
Node-Selector:  <none>
Labels:         name=fluentd
Annotations:    <none>
Desired Number of Nodes Scheduled: 2
Current Number of Nodes Scheduled: 2
Number of Nodes Scheduled with Up-to-date Pods: 2
Number of Nodes Scheduled with Available Pods: 2
Number of Nodes Misscheduled: 0
Pods Status:  2 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:  name=fluentd
  Containers:
   fluentd:
    Image:      gcr.io/google-containers/fluentd-elasticsearch:1.20
    ...
Events:
Type    Reason            Age   From                  Message
  ----    ------            ----  ----                  -------
  Normal  SuccessfulCreate  9m    daemonset-controller  Created pod: fluentd-ds-22jhn
  Normal  SuccessfulCreate  9m    daemonset-controller  Created pod: fluentd-ds-2krwx
```

srinathchalla@outlook.com

DaemonSet – Delete

```
[srinath@master ~]$  kubectl delete ds fluentd-ds
daemonset.extensions "fluentd-ds" deleted
```

```
[srinath@master ~]$ kubectl get po
No resources found.
```

N. Secrets

- Secrets are Kubernetes objects to handle small amount of sensitive data which includes passwords, tokens or keys. Main aim of secrets is to reduce the risk of accidental exposure of confidential information.

- They are created outside the pod and containers. Secrets have no clue on which pod or container it will be used. Once it is created, it can be deployed on any pod any number of times. We do create secrets before they can be used anywhere inside the pod manifest file

- Secrets are stored inside etcd database on Kubernetes master.

- Size of a secret cannot be more than 1MB.

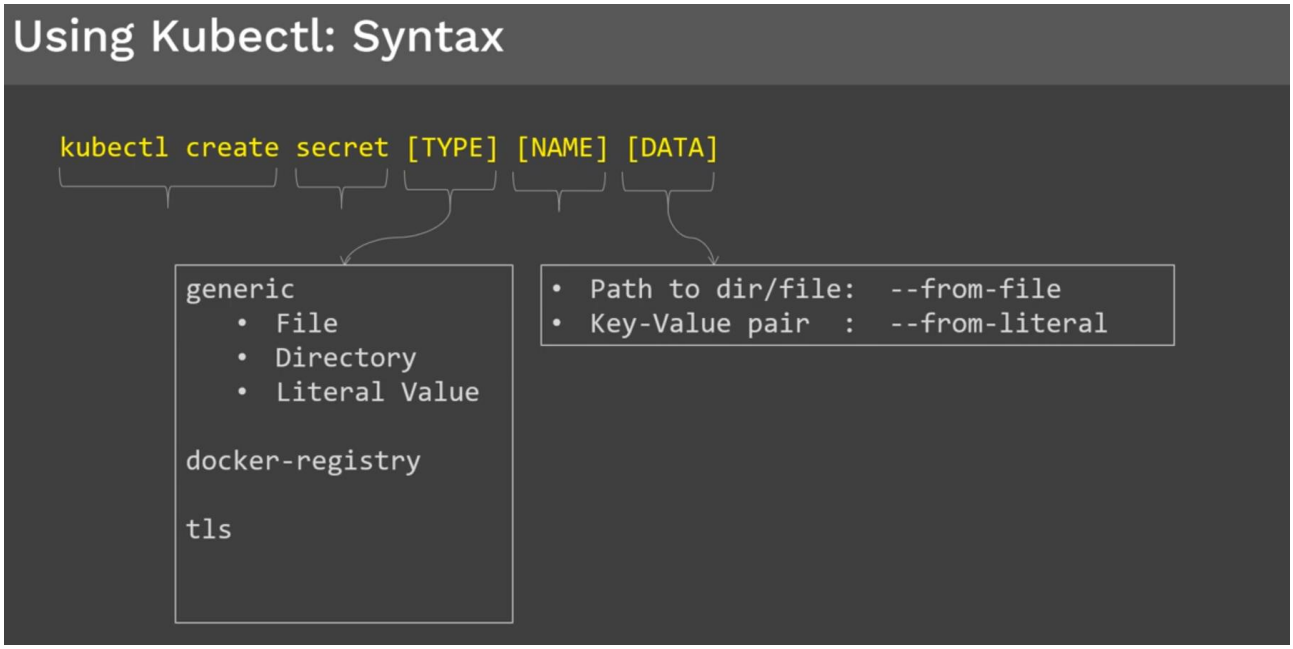
- Used in two ways – mounted as volumes or exposed as env variables.

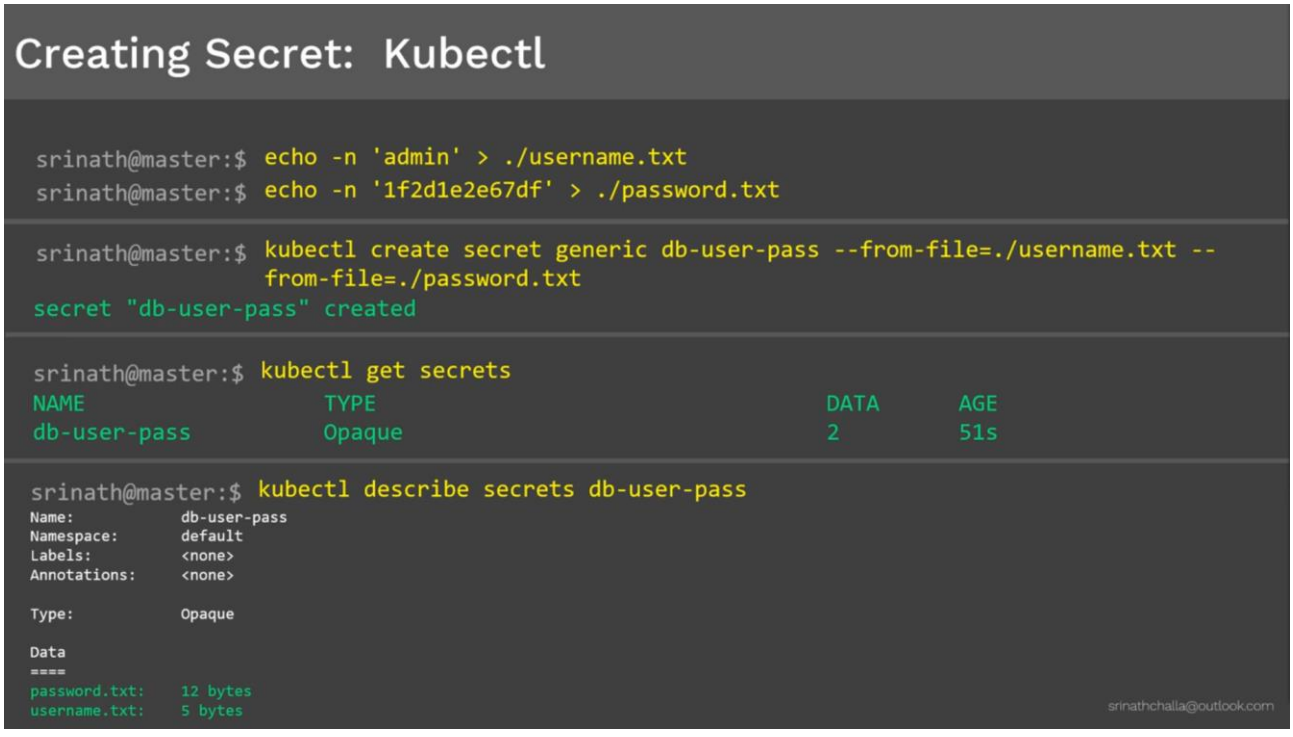
- Sent only to target nodes where pods are running and demand secrets. Unlike other configuration management tools like puppet or ansible where they are broadcasted.

- Each secret is stored in tempfs volume so that access is restricted to the application in the node. An application running in another container cannot access this secret. It is very selective to the pod/container that demands this secret hence it cannot be accidentally revealed.

There are two ways to create secrets:

- Using kubectl
- Manually writing manifest files for secrets





Decoding Secrets

```
srinath@master:$ kubectl get secrets mysecret -o yaml
apiVersion: v1
data:
  password: MWYyZDFlMmU2N2Rm
  username: YWRtaW4=
kind: Secret
metadata:
  creationTimestamp: 2018-09-01T12:46:17Z
  name: mysecret
  namespace: default
  resourceVersion: "616565"
  selfLink:
/api/v1/namespaces/default/secrets/mysecret
  uid: 051e61ae-ade5-11e8-8d64-42010a800003
type: Opaque
```

```
srinath@master:$ echo 'YWRtaW4=' | base64 --decode
admin
srinath@master:$ echo 'MWYyZDFlMmU2N2Rm' | base64 --decode
1f2d1e2e67df
```

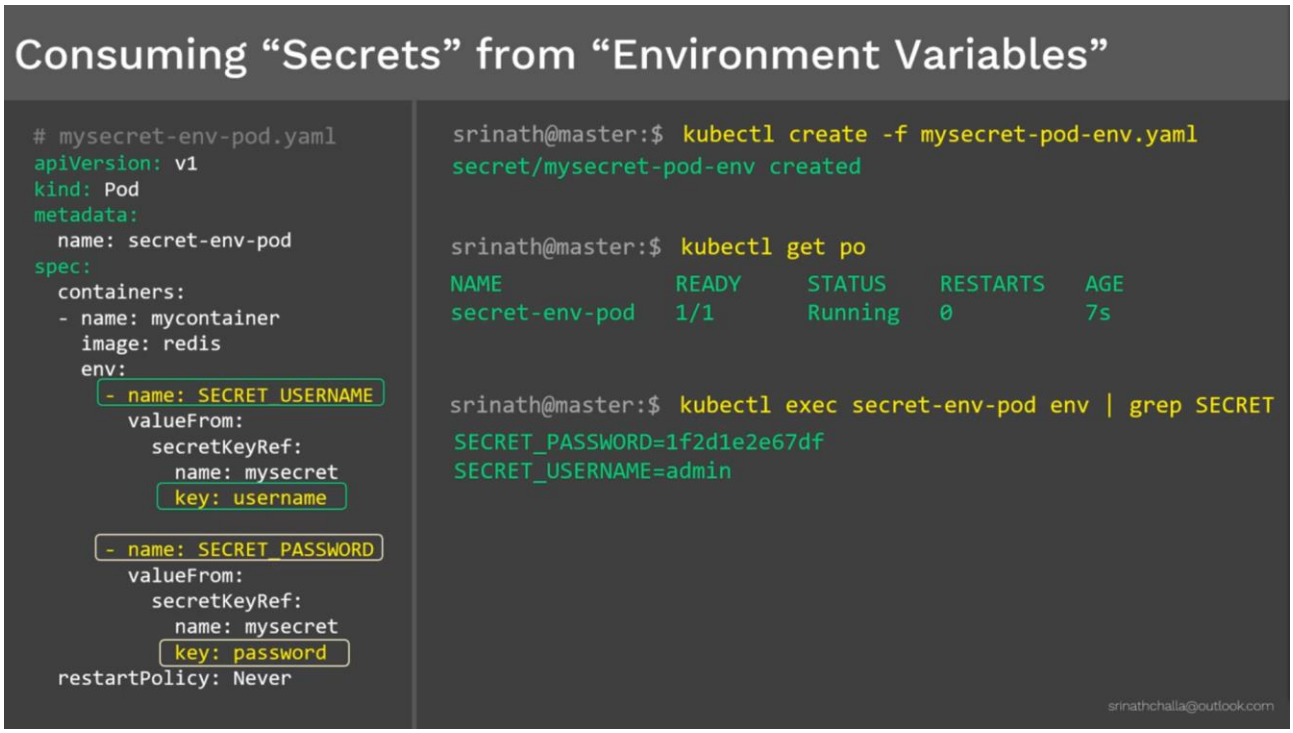
srinathchalla@outlook.com

Consuming “Secrets” from volume

```
# mysecret-pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
  - name: mypod
    image: redis
    volumeMounts:
    - name: foo
      mountPath: "/etc/foo"
      readOnly: true
  volumes:
  - name: foo
    secret:
      secretName: mysecret
```

```
srinath@master:$ kubectl create -f mysecret-pod.yaml
secret/mysecret-pod created

srinath@master:$ kubectl get po
NAME      READY     STATUS              RESTARTS   AGE
mypod     1/1       Running             0          22m

srinath@master:$ kubectl exec mypod ls /etc/foo
password
username

srinath@master:$ kubectl exec mypod cat /etc/foo/passwd
1f2d1e2e67df

srinath@master:$ kubectl exec mypod cat /etc/foo/username
admin
```

srinathchalla@outlook.com



O. Jobs

Job is a controller that supervises pods for carrying out certain tasks.

Two types of jobs:

1. Run to completion (Jobs) : Primarily used for performing batch processing. Once the job manifest file is submitted to the API server, a pod gets created and it will execute a task. And when the task is completed, it will shutdown by itself. The moment a pod gets exit port 0, then job will move away. Once job is completed, pods will move from running state to shutdown state. Kubernetes won’t automatically delete these pods, we have to do it manually. Pods in shutdown state do not consume any resources as there is no more life to it.

2. Scheduled (CronJobs): Purpose of cronjob is to perform cleanup and free disk space and allocate it after a particular interval of time.

Run to completion jobs:

- Each job creates one or more pods.
- Ensure they are successfully terminated.
- Job controller restarts or gets rescheduled if a pod or node fails during execution.
- Can run multiple pods in parallel.
- Can scale up using scale command

Use cases:

- One time initialization of resources such as databases.
- Multiple workers to process messages in queue.

Demo:

Config file

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: countdown
spec:
  template:
    metadata:
      name: countdown
    spec:
      containers:
      - name: counter
        image: centos:7
        command:
         - "bin/bash"
         - "-c"
         - "for i in 9 8 7 6 5 4 3 2 1 ; do echo $i ; done"
      restartPolicy: Never
```

Create, Display and Test

```
[srinath@master ~]$  kubectl create -f countdown-jobs.yaml
job.batch/countdown created
```

```
[srinath@master ~]$ kubectl get jobs
NAME        DESIRED   SUCCESSFUL   AGE
countdown   1         1            11m
```

```
[srinath@master ~]$ kubectl get po
NAME              READY     STATUS      RESTARTS   AGE
countdown-brzdt   0/1       Completed   0          10s
```

```
[srinath@master ~]$ kubectl logs countdown-brzdt
9
8
7
6
5
4
3
2
1
```

Describe

```
[srinath@master ~]$  kubectl describe jobs countdown
Name:           countdown
Namespace:      default
Selector:       controller-uid=5071e78a-ac4e-11e8-b3de-42010a800003
Labels:         controller-uid=5071e78a-ac4e-11e8-b3de-42010a800003
                job-name=countdown
Annotations:    <none>
Parallelism:    1
Completions:    1
Start Time:     Thu, 30 Aug 2018 12:14:59 +0000
Pods Statuses:  0 Running / 1 Succeeded / 0 Failed
Pod Template:
  Labels:  controller-uid=5071e78a-ac4e-11e8-b3de-42010a800003
           job-name=countdown
  Containers:
  ...
  ...
Events:
  Type    Reason            Age   From            Message
  ----    ------            ----  ----            -------
  Normal  SuccessfulCreate  11m   job-controller  Created pod: countdown-brzdt
```


Cleanup

```
[srinath@master ~]$  kubectl delete jobs countdown
job.batch "countdown" deleted
```

```
[srinath@master ~]$ kubectl get po
No resources found
```